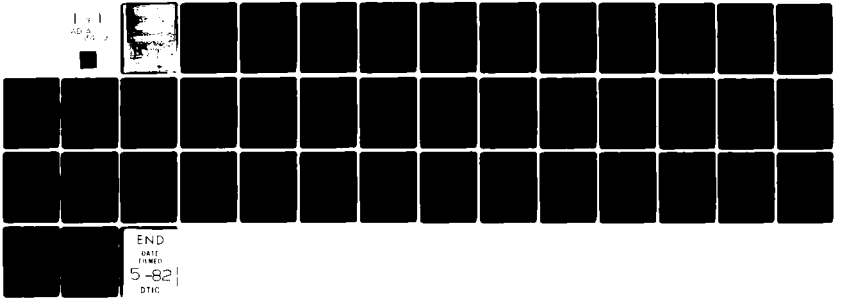
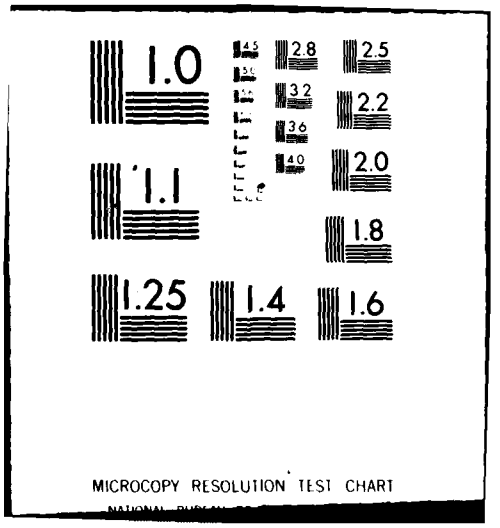
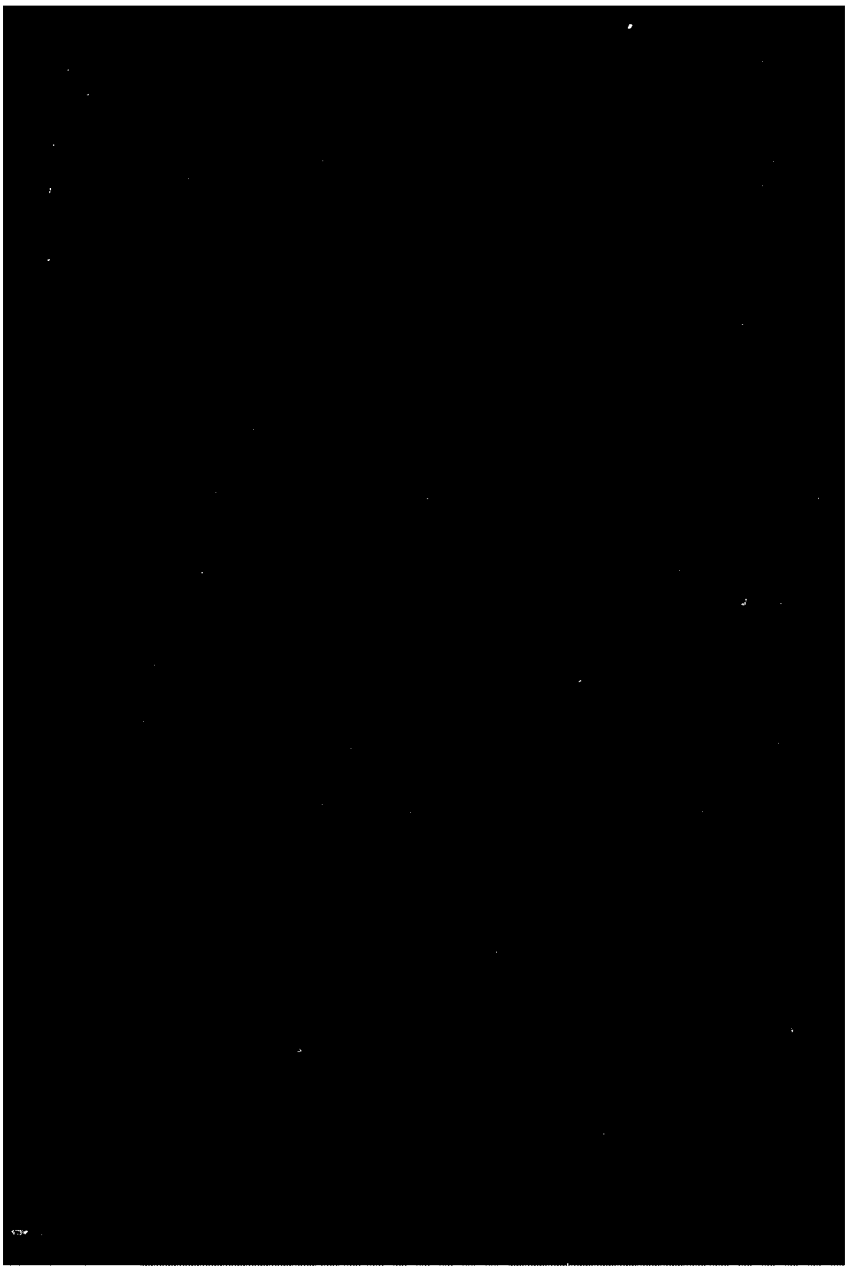
AD-A113 429 DEFENCE RESEARCH ESTABLISHMENT OTTAWA (ONTARIO) F/G 9/2
PROGRAMMING GUIDE FOR COLOUR GRAPHICS ON THE NORPAK VDP/VGS AND--ETC(U)
DEC 81 B J FORD
UNCLASSIFIED DREO-TN-81-25 NL



END
DATE FILMED
5-82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART



(3)

RESEARCH AND DEVELOPMENT BRANCH

DEPARTMENT OF NATIONAL DEFENCE
CANADA

DEFENCE RESEARCH ESTABLISHMENT OTTAWA

TECHNICAL NOTE NO. 81-25

PROGRAMMING GUIDE FOR COLOUR GRAPHICS ON THE NORPAK VDP/VGS AND VDP/RGS

by
Barbara J. Ford
Electronic Warfare Division

PROJECT NO.
31B20

DECEMBER 1981
OTTAWA

ABSTRACT

This paper describes the library of routines written in C language under UNIX on a PDP-11/34 for operating the NORPAK VDP/RGS (raster) and VDP/VGS (vector) colour graphic displays. The details of the routines which are pertinent to the user are described. A listing of the library of routines is included.

RÉSUMÉ

Ce document décrit une bibliothèque de programmes écrit dans le langage C du système UNIX PDP-11/34 qui permet le fonctionnement des écrans couleur Norpak VDP/RGS (trame) et VDP/VGS (vecteur) pour représentation graphique. Les détails des programmes propres à l'utilisateur ainsi qu'une liste de la bibliothèque de programmes y sont inclus.

TABLE OF CONTENTS

PAGE

ABSTRACT/RÉSUMÉ iii

TABLE OF CONTENTS v

1.0 INTRODUCTION 1

2.0 THE PHASES 1
- 2.1 Initialization Phase 1
- 2.2 Subpicture Definition Phase 2
- 2.3 Group Definition Phase 2
- 2.4 Display Phase 2

3.0 THE COMMANDS 3
- 3.1 Control Commands 3
- 3.2 Subpicture and Group Definition Drawing Commands 5
- 3.3 Group Edit Commands 7

4.0 CONCLUSIONS 8

5.0 REFERENCES 9

APPENDIX A - THE LIBRARY OF DISPLAY ROUTINES 11

1.0 INTRODUCTION

The graphics display library is a series of routines that provide a means of defining what will be on the colour display (the Visual Data Processor (VDP) controlling a Raster Graphics Subsystem (RGS) and Vector Graphics Subsystem (VGS)). The routines are written in the C language running under UNIX on a PDP-11/34 computer. When a program to create a display is compiled, this library of display routines is appended to the main program. This display program would have a number of calls to a subset of these library routines.

The set up of the system of display routines was developed to improve the speed of response time over the NORPAK-supplied routines which were TELIDON oriented. These routines had no optimization for the high resolution Visual Data Processor (VDP), were very general, and used a very slow processor.

The graphics display library routines were patterned after a similar set written in assembler for the CANEWS project implemented on a UYK-20 computer. A different host processor (PDP-11/34) necessitated design changes. Although the code for the UYK-20 routines was not used, much of the philosophy is the same. The design was extended to allow the same routines to be used on the VGS (Vector Graphics Subsystem) or the RGS (Raster Graphics Subsystem).

The library of routines allows a large portion of the display contents to be predefined and display changes to be made quickly. This is because the graphical definitions reside in the VDP, waiting for portions to be selected for display. This minimizes the amount of data flowing between the host processor and the VDP.

This programming guide assumes a knowledge of the UNIX operating system and the C language.

2.0 THE PHASES

The graphics display library (gdlib.c) is divided into four phases:

1) initialization phase

2) subpicture definition phase

3) group definition phase

4) display phase

Programs accessing the graphics display library must acknowledge each of these phases.

2.1 Initialization Phase

The initialization phase conists of a call to gdinit( ). This routine initializes variables, the terminal and the character spacing. It is

in this phase that the user decides whether to use the RGS (unit 0) or the VGS (unit 1). The unit value defaults to 0 but may be set to 1 if desired.

2.2 Subpicture Definition Phase

The subpicture definition phase immediately follows the initialization phase. It is not necessary to have any subpictures defined but a call to gdsubend( ) to end the phase is required. There may be many subpictures defined; each one starts with a call to gdsubopn( ) to open the subpicture and ends with a call to gdsubcls( ) to close the subpicture. Once subpictures have been defined they cannot be edited. A subpicture describes a picture element that will likely be repeated more than once in the display. Subpictures would be used mainly to define special symbols, (eg. the symbol for an aircraft).

The body of the subpicture consists of any combination of calls to gdcolr( ), gdints( ), gdblnk( ), gdaset( ), gdcset( ), gdvect( ), gdivec( ), gdtext( ). These routines are definition commands and will be described later. Calls to gdcolr( ) are not recommended on the VGS because considerable time is taken to switch colours. When the VDP displays the screen it sorts by colour then displays all of one colour at a time. However, colours in subpictures can not be accessed to sort.

A subpicture is restricted to 1024, 16 bit words of data in total. The call to gdsubend( ) is required whether there are any subpictures defined or not.

2.3 Group Definition Phase

The group definition phase immediately follows the subpicture definition phase. This phase is terminated by a call to gdgrpend( ). At least one group must be defined. These groups contain the commands that describe what will be on the display.

A group definition consists of zero or more calls to gdcolr( ), gdints( ), gdblnk( ), gdaset( ), gdcset( ); and one or more calls to any one of the three types of drawing commands: gdvect( ) or gdivec( ), gdtext( ), gdsubp( ). A group can have one of the text, vector or subpicture drawing commands but not more than one type. The drawing commands gdvect( ) and gdivec( ) may be intermixed since they are both vector drawing commands even though the latter draws invisible vectors.

The calls to the definition commands simply reset the default attributes assigned to the group. When the gdgrpend( ) call is made the attributes are filled into the group header along with the drawing commands, and output.

2.4 Display Phase

The display phase immediately follows the group definition phase. This phase is terminated by another gdinit( ). If the calling program terminates without another gdinit( ) then the picture will remain displayed

until the screen is accessed by another program.

This phase consists of calls to gdattach( ), gdupdate( ), gdcursor( ), gdpoll( ), gddisply( ) and calls to any of the edit group. Descriptions of all of these routines follow. These calls may be made in any order, depending on application.

3.0 THE COMMANDS

The commands are the actual routines that will be called by the user's program to draw on the display.

3.1 Control Commands

gdinit( ):

This routine sets up a line of communication with the VGS or the RGS. Variables, the terminal and character spacing are initialized.

gdsubopn( ):

This routine indicates a subpicture definition will begin. The routine checks that the phase is correct to open a subpicture, and that there is not an unclosed subpicture. Once a subpicture is defined it cannot be edited. The routine keeps track of the number of subpictures.

gdsubcls( ):

This routine indicates the current subpicture definition will end. The routine checks that the phase is correct to close a subpicture and that there is indeed an open subpicture to close. If so, the subpicture information is written to the VDP.

gdsubend( ):

This routine marks the end of the subpicture definition phase. The routine checks that all subpictures that have been opened have been closed. If not, an error is indicated. If the state is correct to end the subpicture phase it is changed to group phase and the group header information is set up so the group definition phase may begin. Initially the group colour is red, the intensity is full, the screen is not blinking, the drawing pointer is set to (0,0), and the small character set is in use.

gdgrpopn( ):

This routine is called to begin definition of a group. It checks that the phase is the group definition phase; and it checks that there is not an unclosed group. The routine keeps track of the number of groups defined. Each group may define text, or vectors (visible and invisible), or subpicture calls; but may not define a combination of these.

gdgrpcls( ):

This routine closes the definition of a group. The routine checks that the phase is correct to close a group definition and that there is indeed an open group to close. If so, the number of drawing commands in this group is recorded, and the group information is sent to the VDP.

gdgrpend( ):

This routine marks the end of the group definition phase. The routine checks that all groups that have been opened have been closed. The phase is checked and if it is correct (phase 3 - group definition phase) then the program may proceed to the display phase.

gdupdate( ):

This routine updates the display with the given list of groups. The groups are presented in an array along with the number of groups included. The information is sent to the VDP. If the program is not in the display phase an error is indicated.

gdpoll( ):

This routine polls for the co-ordinates of the cursor. The cursor is attached to the trackball. The x position and the y position of the cursor are each read into corresponding variables. If the program is not in the display phase at the time of the call the error is indicated.

gddisply( ):

This routine turns the display refresh on or off. The routine is used for the VGS since the vector system constantly refreshes the screen but the RGS has no need to do so. If the program is not in the display phase at the time of the call an error is indicated.

gdcursor( ):

This routine turns the trackball cursor on or off. The program must be in the display phase.

gdattach( ):

A group defines the shape and size of the cursor, and in order to display the cursor that group must be selected for display through the last update command (gdupdate( )). The routine attaches the trackball to the group defining the cursor. The gdaset( ) value in the defining group is the actual position of the cursor. The information is sent to the VDP. The program must be in the display phase.

3.2 Subpicture and Group Definition Drawing Commands

gdcolr( ):

This routine defines the subpicture colour if in the subpicture definition phase, or changes the colour value in the group header if in the group definition phase. If not in either of these phases an error is indicated. The possible colours are:

0 - red
1 - orange
2 - amber
3 - yellow
4 - green

(Note: the RGS allows variations in blue and purple as well, but as the aim was to make the RGS and VGS work with the same library of routines, the RGS had to limit its colours to those of the VGS.)

gdints( ):

This routine defines the intensity of the information displayed on the screen if in the subpicture phase, or changes the intensity value in the group header if in the group definition phase. If not in either of these phases an error is indicated. The possible intensities are 0 to 17 octal with 17 being full intensity and 0 being invisible. The parameter passed to this routine should contain the intensity.

gdblnk( ):

This routine indicates whether or not the subpicture elements will blink if in the subpicture definition phase; or the routine changes the blink status in the group header if in the group definition phase. If not in either phase an error is indicated. The blink status:

0 - do not blink
1 - blink

gdaset( ):

This routine sets up the absolute x, y position from which the next information will be drawn. In the subpicture definition phase the x, y set is defined. In the group definition phase the x, y set is modified in the header. If in neither of the phases an error is indicated. x and y must be greater than or equal 0, and x and y must be less than or equal 1023. Each group can only do one absolute position set. If a group does not do an absolute position set then the set from the immediately preceding group is used.

gdcset( ):

This routine defines the subpicture character set choice if in the subpicture definition phase, or modifies the character set choice if in the group definition phase. If in neither of these phases an error is indicated. The character set choice is:

0 - small
1 - large

gdvect( ):

This routine sets up a series of one or more relative vectors to be drawn one after the other. The array of vectors may be introduced in either a subpicture or a group. If the program is neither in the subpicture definition phase nor the group definition phase an error is indicated. The array of vectors and the size of the array must be given.

gdivec( ):

This routine sets up a series of one or more invisible relative vectors to be drawn one after another. The array of vectors may be introduced in either a subpicture or a group. If the program is currently neither in the subpicture definition phase nor in the group definition phase an error is indicated. The array of vectors and the size of the array must be given. Invisible vectors may be used if one requires the x, y pointer position to be at another location on the screen but does not wish to start another group.

gdtext( ):

This routine defines a text string to be written on the display, in either a subpicture or a group. If the program is neither in the subpicture definition phase nor in the group definition phase an error is indicated. The string of characters will be sent as follows: gdtext ("ABC", 3). The string and the length of the string (including blanks) must be indicated. Note: the string can also be given in an array.

gdsubp( ):

This routine indicates which subpicture will be accessed from the present group. If the program is not in the group definition phase an error is indicated. As subpictures were defined they were numbered, starting at 1. The subpictures to be accessed are to be indicated using these numbers. These numbers are introduced in the group as an array. The size of this array must also be given.

3.3 Group Edit Commands

After the screen has been displayed portions may need changing. This is done by editing the desired groups and then redisplaying the screen using the gdupdate( ) command. In each edit command the group to be edited must be indicated, along with the new value(s) of what is being edited. Groups were numbered from 1 as they were created. They are identified using this number. The edit type must also be given:

0 - if the template is to be edited
1 - if the template and the display are to be edited

When the display is edited (edit type = 1) an automatic update of the display occurs.

gdedcolr( ):

This routine edits a group's colour.

gdedints( ):

This routine edits a group's intensity.

gdedblnk( ):

This routine edits whether or not the information in the group will blink.

gdedaset( ):

This routine edits the group's absolute x, y pointer position.

gdedcset( ):

This routine edits the group's choice of character size.

gdedvect( ):

This routine edits a number of vectors (0 or more) in a given group. The vectors to be modified are presented in an array. The size of this array must be given. Each array must be of contiguous vectors. The number of the first vector to be changed must be given. There may be more than one call to gdedvect( ) for a given group. This may be necessary since all vectors to be changed are not likely contiguous.

gdedivec( ):

This routine is like gdedvect( ) but the vectors are invisible vectors.

gdedtext( ):

This routine edits a series of contiguous characters of a text string. The number of the first character of the series must be given to the routine along with the number of characters in the series to be changed. Of course the new string of characters must also be given. Since the text edit is a one for one character replacement the new string must be the same length as the old string. There may be more than one gdedtext( ) call per group.

gdedsubp( ):

This routine edits the subpictures to be called from the group. The subpictures are presented in an array. The size of the array must be given. The subpictures to be edited must be contiguous, and the number of the first subpicture of the set must be given. There may be more than one gdedsubp( ) call per group.

4.0 CONCLUSIONS

Appendix A is a complete listing of the library of display routines.

Although the display routines do not include specialized routines such as a routine to draw dashed lines and a routine to draw a circle, the routines are simple, straightforward and easy to use. Using the routines in the library, it is a straightforward matter to write the specialized routines. Once these routines are written they can be compiled along with the calling program or added to the display library.

Consistency among the library routines was emphasized. For example all information for vectors, groups, strings, and subpictures must be presented as arrays.

It is beneficial that the same routines can be used to display on the raster CRT or the vector CRT, with the differences in manipulating the two transparent to the user.

The library of routines provides an effective base for display depiction.

5.0 REFERENCES

1. IDS-VGS Programming Guide, Number D2389, NORPAK Limited.

APPENDIX A

THE LIBRARY OF DISPLAY ROUTINES

```
/* The global variables. */

int gdphase,              /* Keeps track of the phase:
                               1 - initialization.
                               2 - subpicture definition.
                               3 - group definition.
                               4 - display.                    */

    gdbuff[1024],         /* The buffer that stores all of the
                             subpicture group, header, and
                             display information before it is
                             sent to the VDP.                  */

    *gdobptr,             /* The pointer into the output buffer
                             which stores information about the
                             group.                            */

    gdfid[2],             /* When a file is opened it is given
                             an id.                            */

    gddrcomcnt,           /* Stores the drawing command count
                             before it is put into the group
                             header.                           */

    gdspicno,             /* Indicates what subpicture number has
                             just been created. (Initially 1)  */

    gdspicph,             /* 0 - indicates all subpictures that
                                 had been opened have been
                                 closed.
                             1 - indicates there remains an open
                                 subpicture.                   */

    gdgrpno,              /* Indicates what group number has just
                             been created. (Initially 1)       */

    gdgrpph,              /* 0 - indicates all groups that had
                                 been opened have been closed.
                             1 - indicates there remains an open
```

```
                                        group.                          */

    unit;                       /* Logical unit 0 - VGS, 1 - RGS.      */
                                /* Default is 0 - VDP.  If VGS is
                                     wanted do the folowing:
                                     unit = 1;
                                     gdinit();                          */

/*************************************************************************/


vwrite()

/* This routine sets up the write to the VDP. */

{   int bfindx;             /* This stores the size of the used part of the
                                buffer.                                 */

    bfindx = gdobptr - gdbuff;
    if (unit == 0)
    {   if (write(gdfid[1], &bfindx,2) != 2)
        {   printf("VDP write error.\n");
            exit(-1);
        }
    }
    if (write(gdfid[1], gdbuff, bfindx * 2) < 0)
    {   printf("VDP write error.\n");
        exit(-1);
    }
    bfindx = 0;
    gdobptr = gdbuff;
}


/*************************************************************************/


extern errno;
```

```
vread()

/* This routine sets up the read from the VDP. */

{     int  nbytes;

      if (unit == 7)
      {     *gdobptr++ = 0177777;
            vwrite();
      }
      if ( (nbytes = read(gdfid[0], gdbuff, 4)) != 4)
      {
            printf("VDP read error : %d %d\n", errno, nbytes);
            exit(-1);
      }
}


/*************************************************************************/


gdinit()

/* Please reference "NORPAK/ D2389/ IDS-VGS Programming Guide".

   This routine initializes variables and initializes terminal
     and character spacing.                                          */

{     static int flag;    /* Set on first program call to gdinit.  It keeps
                             track of number calls to gdinit.          */
      int forkid;         /* Id returned from fork call. */
      char arg1[3];       /* Array for argumenttl. */
      char arg2[3];       /* Array for argument 2. */
      int  fid0[2];       /* Temporary fid array.  */
      int  fid1[2];       /* Temporary fid array.  */

      if (flag == 0)      /* First call to init. */
      {     flag++;
```

```
/* Rest of processing is dependant on unit. */

if (unit == 0)      /* VDP. */
{
     /* Set up a pipe to the IDSVDP program. */

     if (pipe(fid0) < 0)
     {    printf("Cannot create pipe.\n");
          exit(-1);
     }
     gdfid[0] = fid0[0]; /* get input fid.   */

     if (pipe(fid1) < 0)
     {    printf("Cannot create pipe.\n");
          exit(-1);
     }
     gdfid[1] = fid1[1]; /* get output fid.  */

     /* Time for a fork. */

     if ((forkid = fork()) < 0)
     {    printf("Cannot fork.\n");
          exit(-1);
     }

     if (forkid == 0)
     {
          /* We are the child process. */
          /* Create argument strings for the execute. */

          arg1[0] = fid1[0] / 10 + '0';
          arg1[1] = fid1[0] % 10 + '0';
          arg1[2] = 0;

          arg2[0] = fid0[1] / 10 + '0';
          arg2[1] = fid0[1] % 10 + '0';
          arg2[2] = 0;
```

```
                /* close excess fids.    */
                close(fid1[1]);
                close(fid0[0]);
                /* Must execute IDSVDP. */

                if (execl("/usr/bin/idsvdp","idsvdp",arg1,arg2,0) < 0)
                {    printf("execute failed\n");
                     exit(-1);
                }
           }
           else
           {
                /* Parent process. */
                sleep(2);

                /* close excess fids.    */
                close(fid1[0]);
                close(fid0[1]);
           }
      }
      else
      {
           /* Setup direct I/O to VGS. */

           if ((gdfid[0] = open("/dev/vgs",2)) < 0)
           {    printf("Cannot open VGS.\n");
                exit(-1);
           }
           gdfid[1] = gdfid[0];
      }
}
gdobptr = gdbuff;
gdspicno = 0;
gdspicph = 0;
gdgrpno = 0;
gdgrpph = 0;
gdphase = 1;

*gdobptr++ = 0100000;    /* Initialize terminal. */
```

```
    *gdobptr++ = 9;          /* Initialize character spacing. */
    *gdobptr++ = 13;
    vwrite();
}


/*************************************************************************/


gdcubopn()

/* This routine checks if the phase and state of the phase are right to
     open a subpicture.  If so gdspicno is incremented.                  */
/* Once a subpicture is defined it cannot be edited.                      */

{   if ((gdphase == 2) || (gdphase == 1))
    {   gdphase = 2;
        if (gdspicph == 0)
        {   gdspicph = 1;
            gdspicno =+ 1;
            gdbuff[0] = 0100002;
            gdobptr = &gdbuff[2];
        }
        else
        {   printf("There remains an open subpicture in attempt to open\n");
            printf("a subpicture %d.\n", gdspicno);
        }
    }
    else printf("Wrong phase when attempting to open a subpicture %d.\n",
                  gdspicno);
}


/*************************************************************************/


gdsubcls()

/* This routine checks if the phase and state are right to close a
```

```
     subpicture.  If so the subpicture information is written to the VDP.  */

{    if (gdphase == 2)
     {    if (gdspicph == 1)
          {    gdspicph = 0;
               gdbuff[1] = (gdobptr - &gdbuff[2]);
               vwrite();
          }
          else
          {    printf("All subpictures already closed when attempting to\n");
               printf("close a subpicture %d.\n", gdspicno);
          }
     }
     else printf("Wrong phase to try to close a subpicture %d.\n", gdspicno);
}


/*****************************************************************************/


gdsubend()

/* This routine checks if the phase and state are right to end the
     subpicture phase.  If so the phase is changed to group phase and the
     group header information is set up.  This call must be included
     even if there are no subpictures.                                    */

{    if ((gdphase == 2) || (gdphase == 1))
     {    gdphase = 2;
          if (gdspicph == 0)
          {    gdphase = 3;
               gdbuff[2] = 0;          /* Red. */
               gdbuff[3] = 15;         /* Full intensity. */
               gdbuff[4] = 0;          /* Not blinking. */
               gdbuff[5] = 0;          /* x = 0 */
               gdbuff[6] = 0;          /* y = 0 */
               gdbuff[7] = 0;          /* First character set. */
          }
          else
```

```
        {   printf("There remains an open subpicture when trying to end\n");
            printf("the phase.\n");
        }
    }
    else printf("Wrong phase to end subpicture phase.\n");
}


/*************************************************************************/


gdgrpopn()

/* This routine checks if the phase and state are right to open a group.  If
    so, the gdgrpno is incremented, the drawing command count is initialized,
    the first word of the buffer is set to the group opcode, and the buffer
    pointer is set to after the header information.                        */
/* NOTE:  Can only have text or vectors or subpictures in a group
    and not a combination.                                                */

{   if (gdphase == 3)
    {   if (gdgrpph == 0)
        {   gdgrpno =+ 1;
            gdgrpph = 1;
            gddrcomcnt = 0;
            gdbuff[0] = 0100001;
        }
        else
        {   printf("There remains an open group in attempt to create\n");
            printf("a group %d.\n", gdgrpno);
        }
    }
    else printf("Wrong phase to try to open a group %d.\n", gdgrpno);
}


/*************************************************************************/
```

```
gdgrpcls()

/* This routine checks if the phase and state are right to close a group.
     If so the drawing command count is inserted into the group header
     and the group information is sent to the VDP.                        */

{    if (gdphase == 3)
     {    if (gdgrpph == 1)
          {    gdbuff[1] = gddrcomcnt;
               gdgrpph = 0;
               vwrite();
          }
          else
          {    printf("All groups already closed when attempting to close\n");
               printf("a group %d.\n", gdgrpno);
          }
     }
     else printf("Wrong phase to try to close a group %d.\n", gdgrpno);
}


/*************************************************************************/


gdgrpend()

/* This routine checks if the phase and state are right to end the group
     phase.  If so the phase is changed to display phase.                 */

{    if (gdphase == 3)
     {    if(gdgrpph == 0) gdphase = 4;
          else
          {    printf("There remains an open group when trying to end the\n");
               printf("phase.\n");
          }
     }
     else printf("Wrong phase to end group phase.\n");
}
```

```
/*****************************************************************************/

gdupdate(grparay, grpcnt)

/* This routine indicates a new set of groups are to be displayed.  (Must
     be in phase 4.)  (This information is sent to the VDP.)                 */

int grpcnt,          /* The size of the group array. */

    grparay[];       /* The array of new groups. */

{    int count;      /* A counter to go from 0 to the size of the group
                        array.                                               */

     if (gdphase == 4)
     {    *gdobptr++ = 0100004;
          *gdobptr++ = grpcnt;
          for (count = 0; count < grpcnt; count++)
               *gdobptr++ = grparay[count];
          vwrite();
     }
     else printf("Wrong phase to try gdupdate.\n");
}

/*****************************************************************************/

gdpoll(x, y)

/* This routine reads the position of the cursor into the address of x and
     y.  (Must be in phase 4.)                                               */

int *x,          /* The x position of the cursor. */
    *y;          /* The y position of the cursor. */

{    if (gdphase == 4)
```

```
    {   vread();
        *x = gdbuff[0];
        *y = gdbuff[1];
    }
    else printf("Wrong phase to try gdpoll.\n");
}


/*************************************************************************/


gddisply(flag)

/* This routine turns the display refresh on or off.  This routine is only
    used for the VGS.  (Must be in phase 4.)                              */

int flag;       /* If flag = 0 then the display is refreshed.
                   If flag = 1 then the display refresh is inhibited. */

{   if (gdphase == 4)
    {   *gdobptr++ = 0100007;
        *gdobptr++ = flag;
        vwrite();
    }
    else printf("Wrong phase to try gddisply.\n");
}


/*************************************************************************/


gdcursor(flag)

/* This routine turns the cursor on or off.  When the cursor is on, the
    trackball is attached.  (The information is sent to the VDP.) (Must
    be in phase 4.)                                                       */

int flag;       /* If flag = 0 the cursor is off.
                   If flag = 1 the cursor is on.   */
```

```
{     if (gdphase == 4)
      {     *gdobptr++ = 0100006;
            *gdobptr++ = flag;
            vwrite();
      }
      else printf("Wrong phase to try gdcursor.\n");
}


/*************************************************************************/


gdattach(groupno)

/* This routine attaches the trackball to the set position of the indicated
     group.  (The information is sent to the VDP.)  (Must be in phase 4.)  */

int groupno;    /* The indicated group number. */

{     if (gdphase == 4)
      {     *gdobptr++ = 0100003;
            *gdobptr++ = groupno;
            vwrite();
      }
      else printf("Wrong phase to try gdattach.\n");
}


/*************************************************************************/


gdcolr(colour)

/* This routine sets up the subpicture colour if in phase 2 or changes the
     colour in the group header if in phase 3.  The colours are:
     0 - red;  1 - orange;  2 - amber;  3 - yellow;  4 - green.           */

int colour;          /* The colour to be used. */
```

```
{     if (gdphase == 2)
      {     *gdobptr++ = 0100020;
            *gdobptr++ = colour;
      }
      else if (gdphase == 3)
                  gdbuff[2] = colour;
            else printf("Wrong phase to try gdcolr.\n");
}


/*****************************************************************************/


gdints(intens)

/* This routine sets up the subpicture intensity if in phase 2 or changes
     the intensity in the group header if in phase 3.                        */

int intens;          /* The intensity to be used. (0 to 17 octal) */

{     if (gdphase == 2)
      {     *gdobptr++ = 0100021;
            *gdobptr++ = intens;
      }
      else if (gdphase == 3)
                  gdbuff[3] = intens;
            else printf("Wrong phase to try gdints.\n");
}


/*****************************************************************************/


gdblnk(blnkst)

/* This routine sets up the blink status of the subpicture in phase 2 or
     changes the blink status in the group header if in phase 3.             */
```

```
int blnkst;          /* The blink status. (0 - off, 1 - on) */

{    if (gdphase == 2)
     {    *gdobptr++ = 0100022;
          *gdobptr++ = blnkst;
     }
     else if (gdphase == 3)
               gdbuff[4] = blnkst;
          else printf("Wrong phase to try gdints.\n");
}


/*************************************************************************/


gdaset(absx, absy)

/* The absolute x, y position set is set up in the subpicture if in phase
     2 or changed in the header if in phase 3.                          */
/* NOTE:  Can only have one gdaset per group.                            */
/* NOTE:  If a group does not set an x and y position, the aset from the
          previous group is used.                                       */

int absx,            /* The absolute x position. */
    absy;            /* The absolute y position. */

{    if (gdphase == 2)
     {    *gdobptr++ = 0100023;
          *gdobptr++ = absx;
          *gdobptr++ = absy;
     }
     else if (gdphase == 3)
          {    gdbuff[5] = absx;
               gdbuff[6] = absy;
          }
          else printf("Wrong phase to try gdaset.\n");
}
```

```
/*************************************************************************/


gdcset(charset)

/* This routine sets up the subpicture character set choice if in phase 2,
     or changes the character set choice in the group header if in phase 3.*/
/* 0 - small ;   1 - large .                                               */

int charset;        /* The character set choice. */

{    if (gdphase == 2)
     {    *gdobptr++ = 0100024;
          *gdobptr++ = charset;
     }
     else if (gdphase == 3)
               gdbuff[7] = charset;
          else printf("Wrong phase to try gdcset.\n");
}


/*************************************************************************/


gdvect(dxdyaray, nodxdy)

/* This routine sets up a series of vectors (in relative fashion) in either
     the subpicture or the group mode.                                     */

int dxdyaray[],     /* The array of relative points making up the vectors. */

     nodxdy;        /* The number of points in the array. */

{    int sizecnt,        /* This keeps track which dx or dy is currently
                            being handled.                                 */

          temp;          /* This is used for intermediate calculations
                            on the dx or dy values.                        */
```

```
    if (gdphase == 2)
    {    *gdobptr++ = 0100030;
         sizecnt = 0;
         while (sizecnt <= (nodxdy * 2 - 1))
         {    temp = dxdyaray[sizecnt];
              temp = (temp + 1024) & 043777;      /* The dx, dy must be
                                                     positive, this
                                                     equation accomplishes
                                                     this.              */

              *gdobptr++ = temp;
              sizecnt =+ 1;
         }
    }
    else if (gdphase == 3)
         {    if (gddrcomcnt == 0)
              {    gdobptr = gdbuff + 8;
                   *gdobptr++ = 0100030;
              }
              gddrcomcnt =+ nodxdy;
              sizecnt = 0;
              while (sizecnt <= (nodxdy * 2 - 1))
              {    temp = dxdyaray[sizecnt];
                   temp = (temp + 1024) & 2047;
                   *gdobptr++ = temp;
                   sizecnt =+ 1;
              }
         }
         else printf("Wrong phase when trying gdvect.\n");
}

/*************************************************************************/


gdivec(dxdyaray, nodxdy)

/* This routine sets up a series of invisible relative vectors in either the
subpicture or the group mode.                                          */
```

```
int dxdyaray[],     /* The array of relative points making up the vectors. */
    nodxdy;         /* The number of points in the array.   */

{   int sizecnt,        /* This keeps track which dx or dy is currently
                           being handled.                                  */

        temp;           /* This is used for intermediate calculations on
                           the dx or dy values.                            */

    if (gdphase == 2)
    {   sizecnt = 0;
        *gdobptr = 0100030;
        while (sizecnt <= (nodxdy * 2 - 1))
        {   temp = dxdyaray[sizecnt];
            temp = (temp + 1024) & 2047;        /* The dx, dy must be
                                                   positive, this
                                                   equation accomp-
                                                   lishes this.      */

            if ((sizecnt % 2) == 0)
                temp = temp | 040000;           /* To indicate an invisible
                                                   vector set is des-
                                                   ired, bit 14 in the
                                                   dx data field is
                                                   set.              */

            *gdobptr++ = temp;
            sizecnt =+ 1;
        }
    }
    else if (gdphase == 3)
        {   if (gddrcomcnt == 0)
            {   gdobptr = gdbuff + 8;
                *gdobptr++ = 0100030;
            }
            gddrcomcnt =+ nodxdy;
            sizecnt = 0;
            while (sizecnt <= (nodxdy * 2 - 1))
            {   temp = dxdyaray[sizecnt];
                temp = (temp + 1024) & 2047;
                if (( sizecnt % 2) == 0)
```

```
                    temp = temp | 040000;
                *gdobptr++ = temp;
                sizecnt =+ 1;
            }
        }
        else printf("Wrong phase when trying gdivec.\n");
}


/*************************************************************************/


gdtext(chstradr, count)

/* This routine sets up a series of characters in either the subpicture or
     the group mode.  The string will be sent as indicated: eg)
     gdtext("ABC", 3).                                                    */

int count;          /* This is the number of characters in the text string. */

char *chstradr;     /* This contains the text string. */

{    int tempcnt;   /* This is used to keep track of which character is
                       currently being handled.                          */

     if (gdphase == 2)
     {    *gdobptr++ = 0100031;
          for (tempcnt = 1; tempcnt <= count; tempcnt++)
          {    *gdobptr++ = *chstradr++;
          }
     }
     else if (gdphase == 3)
          {    if (gddrcomcnt == 0)
               {    gdobptr = gdbuff + 8;
                    *gdobptr++ = 0100031;
               }
               gddrcomcnt =+ count;
               for (tempcnt = 1; tempcnt <= count; tempcnt++)
               {    *gdobptr++ = *chstradr++;
```

```
            }
        }
        else printf("Wrong phase for gdtext.\n");
}

/*************************************************************************/

gdsubp(subparr, nosubp)

/* This routine will only allow phase 3.  It indicates a given subpicture
     will be called from the group currently being built.                */

int subparr[],      /* This is the array of the subpicture numbers that
                         will be accessed from this group.              */
    nosubp;         /* This is the number of subpictures in the array.  */

{   int tempcnt;    /* This is used to keep track of which subpicture
                         entry is currently being handles.              */
    if (gdphase == 3)
    {   if (gddrcomcnt == 0)
        {   gdobptr = gdbuff + 8;          /* To avoid the header info.  */
            *gdobptr++ = 0100032;
        }
        gddrcomcnt =+ nosubp;
        for(tempcnt = 0; tempcnt < nosubp; tempcnt++)
            *gdobptr++ = subparr[tempcnt];
    }
    else printf("Wrong phase for gdsubp.\n");
}

/*************************************************************************/

gdedcolr(groupno, editype, colour)

/* This routine will only allow phase 4.  It edits the colour in a given
```

```
     group.                                                              */

int groupno,          /* The group number to be edited. */
     editype,         /* An indicator of whether the template (0) or both the
                           template and display (1) are to be edited.    */
                      /* Note:  when display is edited an automatic
                           update of the display occurs.                 */
     colour;          /* The new colour. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100020;
          *gdobptr++ = colour;
          vwrite();
     }
     else printf("Wrong phase to try gdedcolr.\n");
}


/*************************************************************************/


gdedints(groupno, editype, intens)

/* This routine will only allow phase 4.  It edits the intensity in a
     given group.                                                        */

int groupno,          /* The group to be edited. */

     editype,         /* An indicator of whether the template or both the
                           template and display are to be edited.        */

     intens;          /* The new intensity. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
```

```
            *gdobptr++ = editype;
            *gdobptr++ = 0100021;
            *gdobptr++ = intens;
            vwrite();
        }
        else printf("Wrong phase to try gdedints.\n");
}

/*****************************************************************************/

gdedblnk(groupno, editype, blnkst)

/* This routine will only allow phase 4.  It edits the blink status in a
     given group.                                                            */

int groupno,            /* The group to be edited. */

     editype,           /* Template or both template and display to be edited? */

     blnkst;            /* The new blink status. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100022;
          *gdobptr++ = blnkst;
          vwrite();
     }
     else printf("Wrong phase to try gdedblnk.\n");
}

/*****************************************************************************/

gdedaset(groupno, editype, absx, absy)
```

```
/* This routine will only allow phase 4.  It edits the new absolute set of
     the given group.                                                        */

int groupno,          /* The group to be edited. */

     editype,         /* Template (0) or both template and display (1) to
                           be edited?                                        */

     absx,            /* The new absolute x position. */

     absy;            /* The new absolute y position. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100023;
          *gdobptr++ = absx;
          *gdobptr++ = absy;
          vwrite();
     }
     else printf("Wrong phase to try gdedaset.\n");
}

/*************************************************************************/


gdedcset(groupno, editype, charset)

/* This routine will only allow phase 4.  It edits the chosen character
     set of the given group.                                                 */

int groupno,          /* The group to be edited. */

     editype,         /* Template (0) or both template and display (1) to
                           be edited?                                        */
```

```
     charset;       /* New character set choice. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100024;
          *gdobptr++ = charset;
          vwrite();
     }
     else printf("Wrong phase to try gdedcset.\n");
}


/*************************************************************************/


gdedvect(groupno, editype, start, dxdyaray, nodrcom)

/* This routine will only allow phase 4.  The absolute vectors in a given
     group will be edited.  Since it is a drawing command opcode the start
     of the drawing command changes must be included (start);  the number
     of drawing commands to changed must be included (nodrcom);  and the
     array of new dx, dy values must be included (dxdyaray).            */

int groupno, editype, start, nodrcom, dxdyaray[];

{    int count,     /* This indicates which dx or dy we are currently
                       changing.                                        */

          temp;     /* This is used for intermediate calculations on the
                       dx or dy values.                                 */

     if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100030;
          *gdobptr++ = start;
```

```
          *gdobptr++ = nodrcom;
          for (count = 0; count <= (nodrcom * 2 - 1); count++)
          {    temp = dxdyaray[count];
               temp = (temp + 1024) & 2047;
               *gdobptr++ = temp;
          }
          vwrite();
     }
     else printf("Wrong phase to try gdedvect.\n");
}


/*************************************************************************/


gdedivec(groupno, editype, start, dxdyaray, nodrcom)

/* This routine will only allow phase 4.  The relative vectors in a given
     group will be edited.  Since it is a drawing command opcode the
     start of the drawing command changes must be included (start);  the
     number of drawing commands to be changed must be included (nodrcom);
     and the array of new dx, dy values must be included (dxdyaray).      */

int groupno, editype, start, nodrcom, dxdyaray[];

{    int count,     /* This indicates which dx or dy we are currently
                       changing.                                        */

          temp;     /* This is used for intermediate calculations on
                       the dx or dy values.                             */

     if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100030;
          *gdobptr++ = start;
          *gdobptr++ = nodrcom;
          for (count = 0; count <= (nodrcom * 2 - 1); count++)
```

```
        {   temp = dxdyaray[count];
            temp = (temp + 1024) & 2047;
            if ((count & 2) == 0)
                temp = temp | 040000;
            *gdobptr++ = temp;
        }
        vwrite();
    }
    else printf("Wrong phase to try gdedivec.\n");
}


/*************************************************************************/


gdedtext(groupno, editype, start, chstradr, nodrcom)

/* This routine will only allow phase 4.  The text in a given group will be
     edited.  Since it is a drawing command there must:  the start of the
     drawing command changes (start);  the number of drawing commands
     (ie. characters to be changed) (nodrcom); and address of the character
     string (ie. new value) (chstradr).                                    */

int groupno, editype, start, nodrcom;
char *chstradr;

{   int count;          /* This keeps track of which character is
                           currently being handled.          */

    if (gdphase == 4)
    {   *gdobptr++ = 0100005;
        *gdobptr++ = groupno;
        *gdobptr++ = editype;
        *gdobptr++ = 0100031;
        *gdobptr++ = start;
        *gdobptr++ = nodrcom;
        for (count = 1; count <= nodrcom; count++)
        {   *gdobptr++ = *chstradr++;
        }
```

```
        vwrite();
    }
    else printf("Wrong phase to try gdedtext.\n");
}


/*************************************************************************/


gdedsubp(groupno, editype, start, subparay, nodrcom)

/* This routine will only allow phase 4.  The subpicture number to be
    called from the given group will be edited.  Since it is a drawing
    command there must be:  The start of the drawing command changes
    (start);  the number of drawing commands or in this case the
    number of consecutive subpicture calls to be changed (nodrcom);
    and the array of new consecutive subpicture calls (subparay).    */

int groupno, editype, start, nodrcom, subparay[];

{   int count;          /* This keeps track of which subpicture call
                           is currently being changed.              */

    if (gdphase == 4)
    {   *gdobptr++ = 0100005;
        *gdobptr++ = groupno;
        *gdobptr++ = editype;
        *gdobptr++ = 0100032;
        *gdobptr++ = start;
        *gdobptr++ = nodrcom;
        for (count = 0; count <= nodrcom - 1; count++)
            *gdobptr++ = subparay[count];
        vwrite();
    }
    else printf("Wrong phase to try gdedsubp.\n");
}
```

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1 ORIGINATING ACTIVITY
Defence Research Establishment Ottawa
National Defence Headquarters
Ottawa, Ontario K1A 0Z4

2a DOCUMENT SECURITY CLASSIFICATION
Unclassified

2b GROUP

3 DOCUMENT TITLE
Programming Guide for Colour Graphics on the NORPAK VDP/VGS and VDP/RGS (U)

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Note

5. AUTHOR(S) (Last name, first name, middle initial)
Ford, Barbara J.

6. DOCUMENT DATE	7a. TOTAL NO OF PAGES	7b. NO. OF REFS
December	40	1

8a. PROJECT OR GRANT NO.	9a. ORIGINATOR'S DOCUMENT NUMBER(S)
31B20	DREO TN# 81-25
8b CONTRACT NO	9b. OTHER DOCUMENT NO.(S) (Any other numbers that may be assigned this document)

10 DISTRIBUTION STATEMENT
UNLIMITED DISTRIBUTION

11 SUPPLEMENTARY NOTES	12. SPONSORING ACTIVITY
	CRAD

13 ABSTRACT

This paper describes the library of routines written in C language under UNIX on a PDP-11/34 for operating the NORPAK VDP/RGS (raster) and VDP/VGS (vector) colour graphic displays. The details of the routines which are pertinent to the user are described. A listing of the library of routines is included.

DSIS

KEY WORDS

COLOUR GRAPHIC

PROGRAMMING GUIDE

SUBPICTURES

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the organization issuing the document.

2a. DOCUMENT SECURITY CLASSIFICATION: Enter the overall *security classification of the document* including special warning terms whenever applicable.

2b. GROUP: Enter security reclassification group number. The three groups are defined in Appendix 'M' of the DRB Security Regulations.

3. DOCUMENT TITLE: Enter the complete document title in all capital letters. Titles in all cases should be unclassified. If a sufficiently descriptive title cannot be selected without classification, show title classification with the usual one-capital-letter abbreviation in parentheses immediately following the title.

4. DESCRIPTIVE NOTES: Enter the category of document, e.g. technical report, technical note or technical letter. If appropriate, enter the *type of document, e.g. interim, progress,* summary, annual or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the document. Enter last name, first name, middle initial. If military, show rank. The name of the principal author is an absolute minimum requirement.

6. DOCUMENT DATE: *Enter the date (month, year) of* Establishment approval for publication of the document.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the document.

8a. PROJECT OR GRANT NUMBER: If appropriate, enter the applicable research and development project or grant number under which the document was written.

8b. CONTRACT NUMBER: *If appropriate, enter the applicable* number under which the document was written.

9a. ORIGINATOR'S DOCUMENT NUMBER(S): Enter the official document number by which the document will be identified and controlled by the originating activity. This number must be unique to this document.

9b. OTHER DOCUMENT NUMBER(S): If the document has been assigned any other document numbers (either by the originator or by the sponsor), also enter this number(s).

10. DISTRIBUTION STATEMENT: Enter any limitations on further dissemination of the document, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this *document from their defence documentation center.*"

(2) "Announcement and dissemination of this document is not authorized without prior approval from originating activity."

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document, even though it may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall end with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (TS), (S), (C), (R), or (U).

The length of the abstract should be limited to 20 single-spaced standard typewritten lines; 7½ inches long.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a document and could be helpful in cataloging the document. Key words should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context.


DAT
FILM